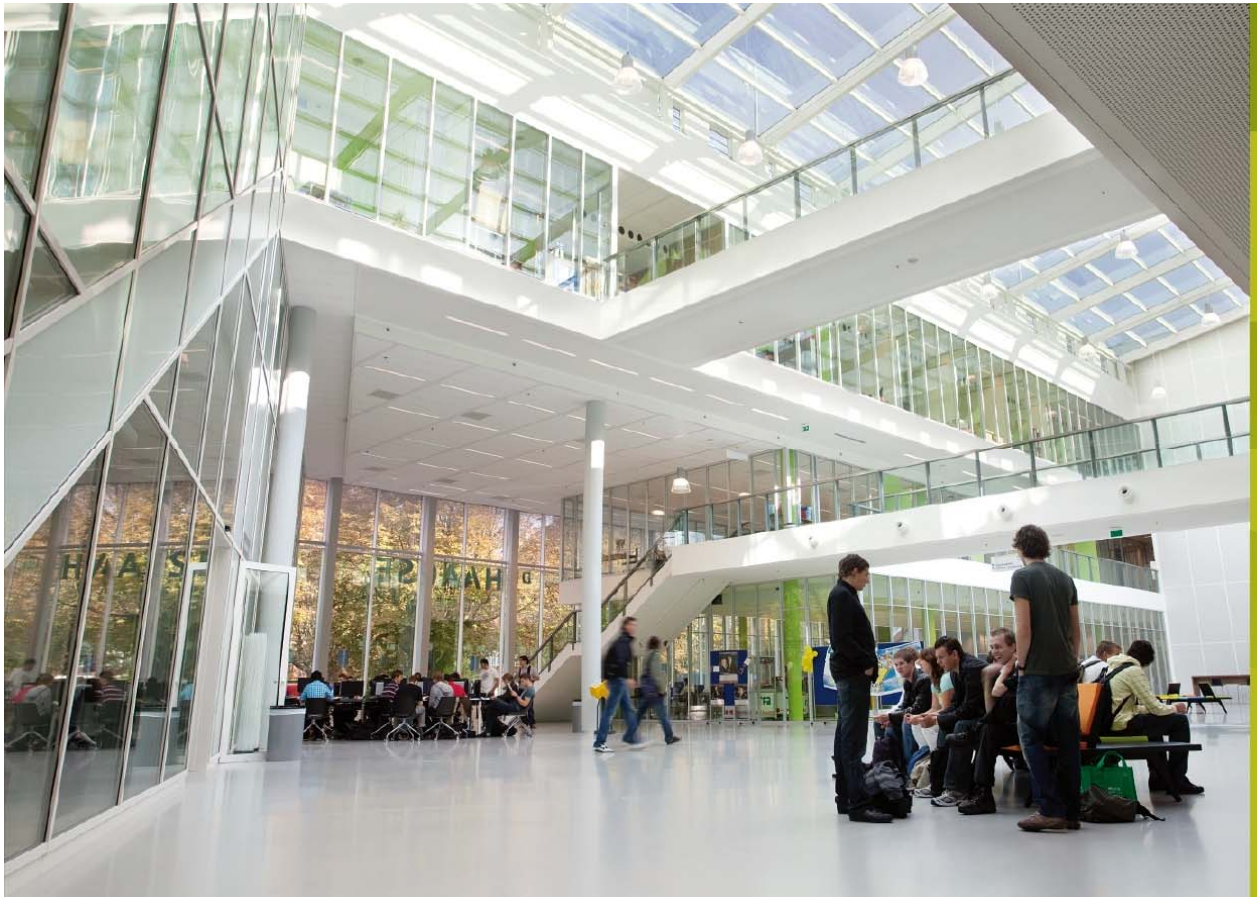


Academie voor Technology, Innovation & Society Delft
Academie voor ICT & Media

Microcontrollers

Week 3 – Introductie microcontroller
Jesse op den Brouw (met dank aan Ben Kuiper)
INLMIC/2018-2019

HAAGSE HOGESCHOOL

Week 3

- Adressing modes
- Indexregisters
- Flags en instructies
- I/O ports
- Bitmaskers
- Vergelijkingen en beslissingen

DE HAAGSE HOGESCHOOL

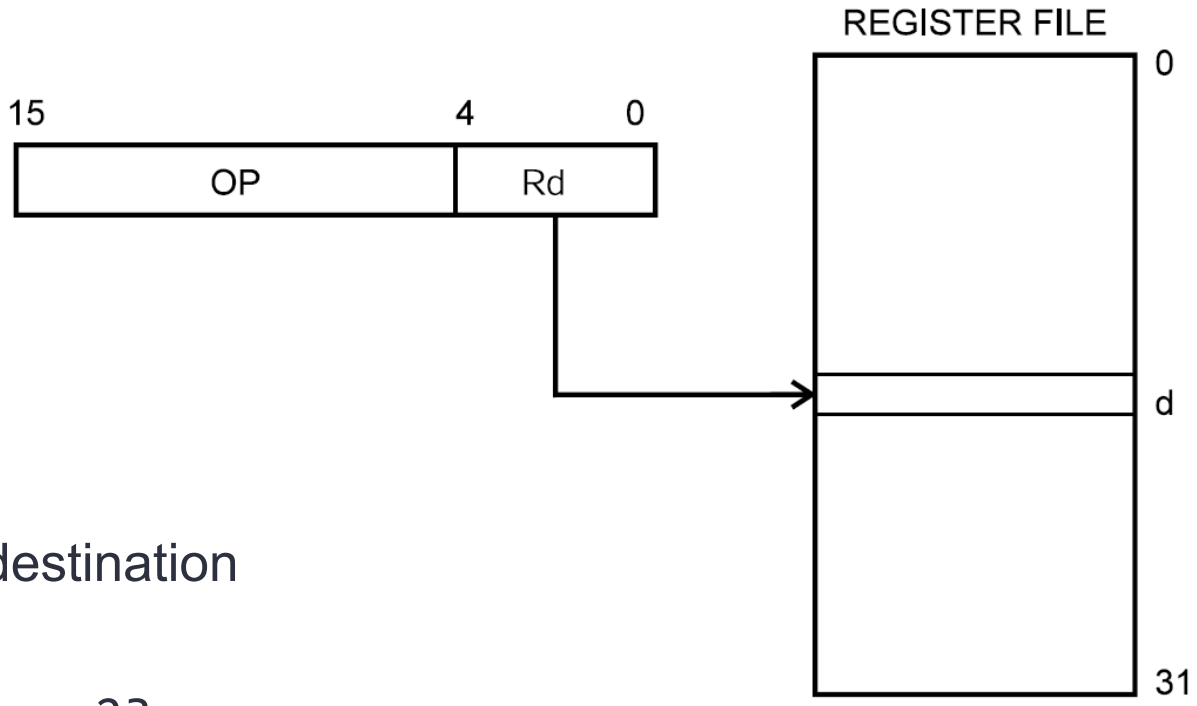
Adressing modes

- De operands zijn te verdelen in:
  - Constante: 0 – 255
  - Register: r0 – r31
  - I/O adres: 0x00 – 0x3f
  - Adres: 0x0000 – 0xffff (65536 plaatsen, 16 bits)
  - Indexregister: X, Y, Z
  - Sprongadres: 0x000000 – 0x3fffff (4M plaatsen, dus 22 bits)*
- Dit worden *addressing modes* (manieren van adresseren) genoemd.
  - noot: lijst is niet volledig
- Sommige instructies hebben geen operands.
- *) ATmega32 gebruikt onderste 14 bits, dus 16k plaatsen

DE HAAGSE HOGESCHOOL

Addressing modes

- Direct Single Register

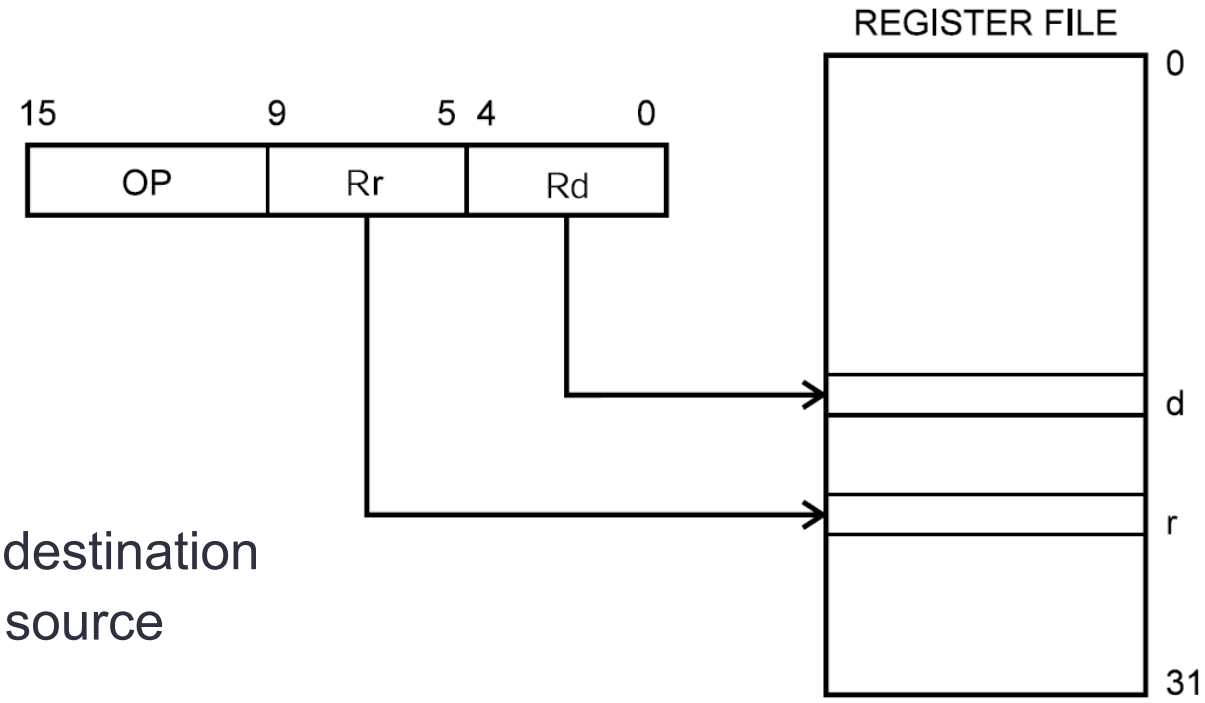


- OP = opcode
- Rd = register destination

- Voorbeeld: `inc r23`

Addressing modes

- Direct Register Addressing, Two Registers

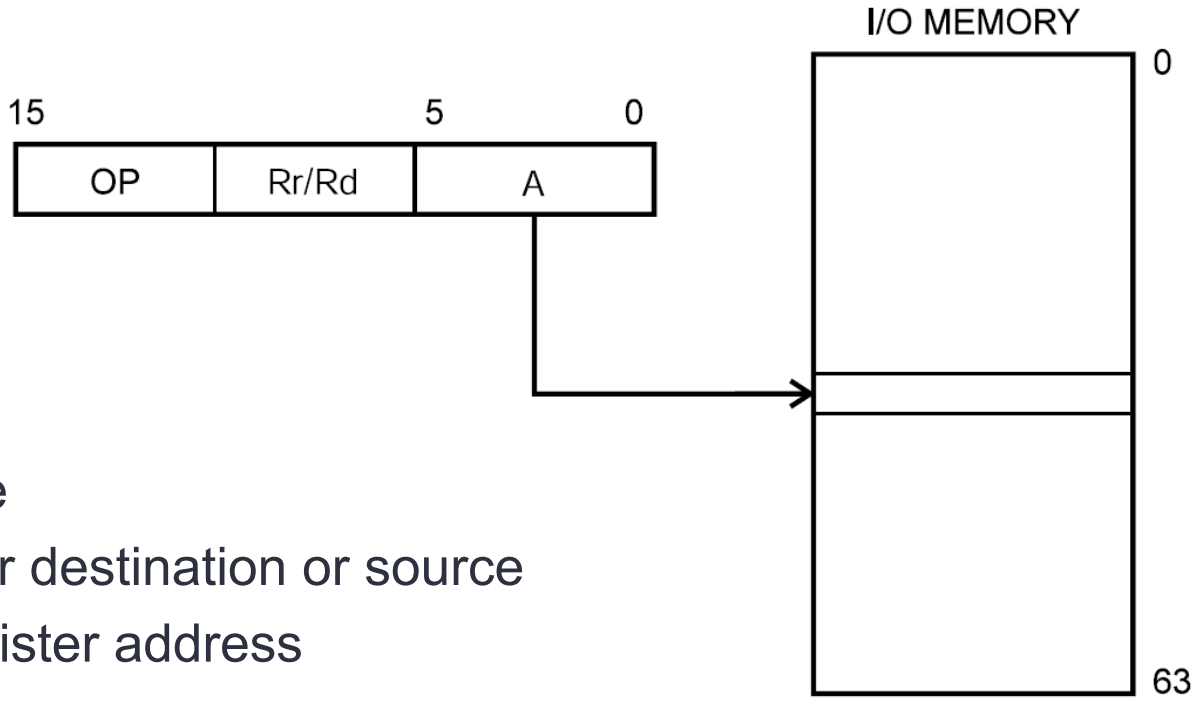


- OP = opcode
- Rd = register destination
- Rr = register source

- Voorbeeld: `add r5,r1`

Addressing modes

- I/O Direct Addressing



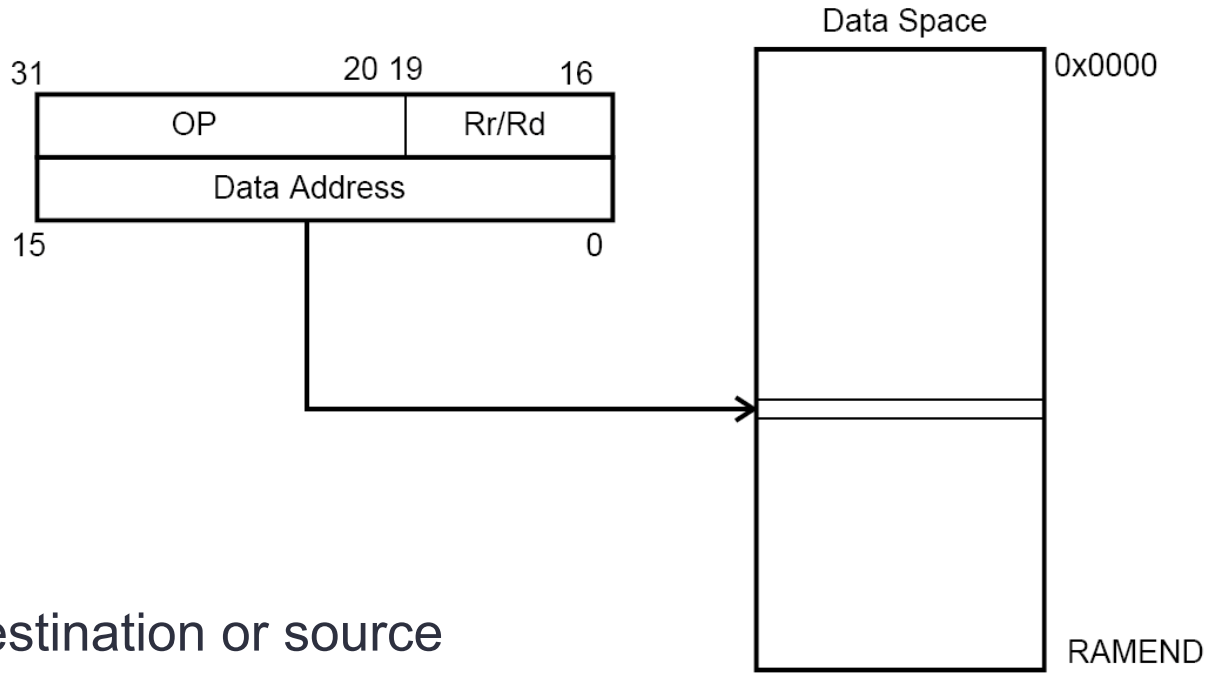
- OP = opcode
- Rd/Rr = register destination or source
- A = I/O register address

- Voorbeeld:

```
out   0x18,r4
in    r5,0x19
```

Addressing modes

- Direct Data Adressing



- OP = opcode
- Rd/Rr = register destination or source

- Voorbeeld:

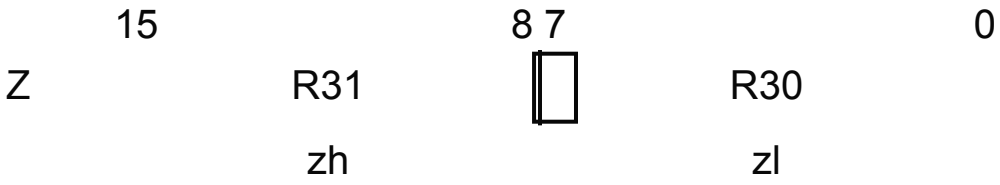
```
lds   r1,0x0060
sts   0x0060,r4
```

Indexregisters

- Tot nu toe hebben we data adressing modes gezien waarbij het adres vast lag, het was niet te veranderen.

- Er zijn ook modes waarbij een register*paar* gebruikt kan worden om een geheugenplaats in RAM aan te wijzen.

- Eén zo'n registerpaar is het Z-register.
- Register R31 en R30 vormen samen Z-indexregister.



Indexregisters

- Zo'n registerpaar is makkelijk aan te passen

```
ldi    r30,0x60      ; load immediate r30 with 0x60
ldi    r31,0x00      ; load immediate r31 with 0x00
; Z-index now points to address 0x0060
adiw   r31:r30,1     ; add 1 to Z-index
ld     r16,Z         ; load R16 with ?
```

- De `adiw`-instructie telt het getal 1 op bij het registerpaar R31 en R30 (het Z-indexregister).

- Na het uitvoeren van die instructie wijst het Z indexregister naar het eerstvolgende adres, dus 0x0061.

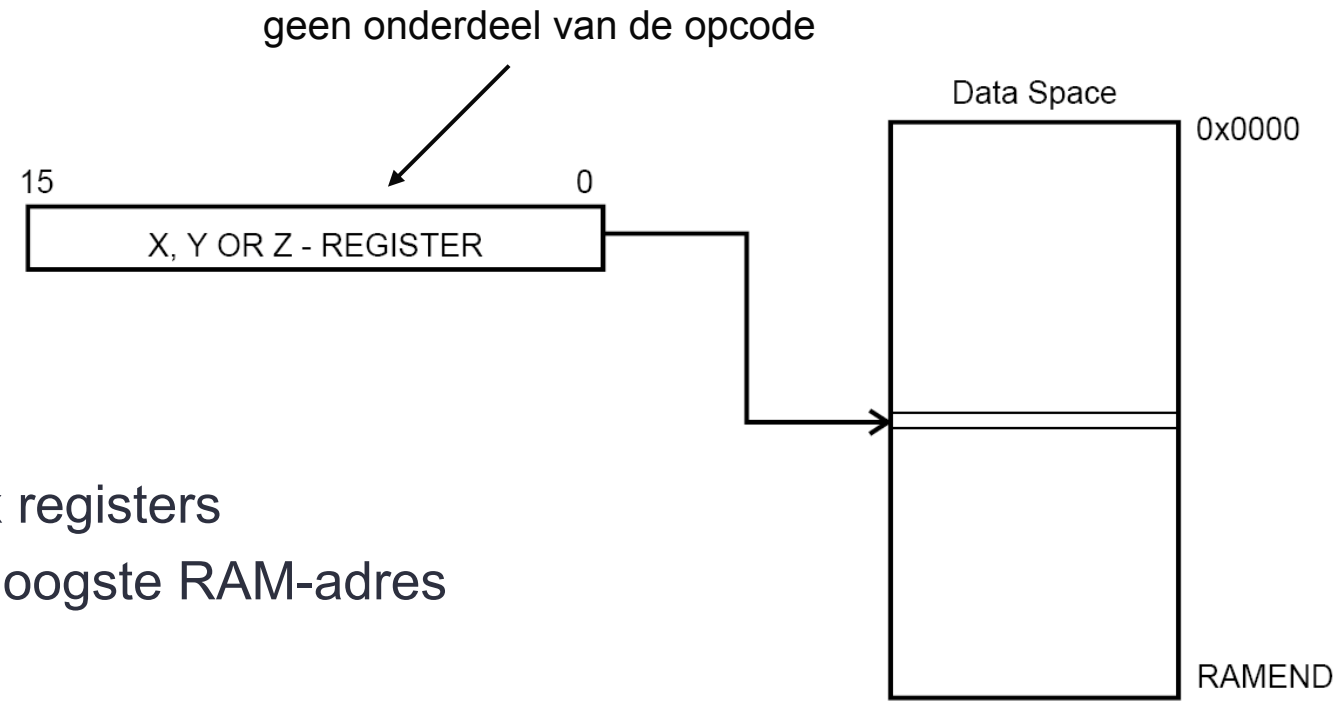
Indexregisters

- Er zijn drie indexregisters:
- R27:R26 vormen het X-indexregister.
- R29:R28 vormen het Y-indexregister.
- R31:R30 vormen het Z-indexregister

- R25:R24 vormt ook een indexregister, maar alleen voor `adiw` en `sbiw`.

Addressing modes

- Data indirect

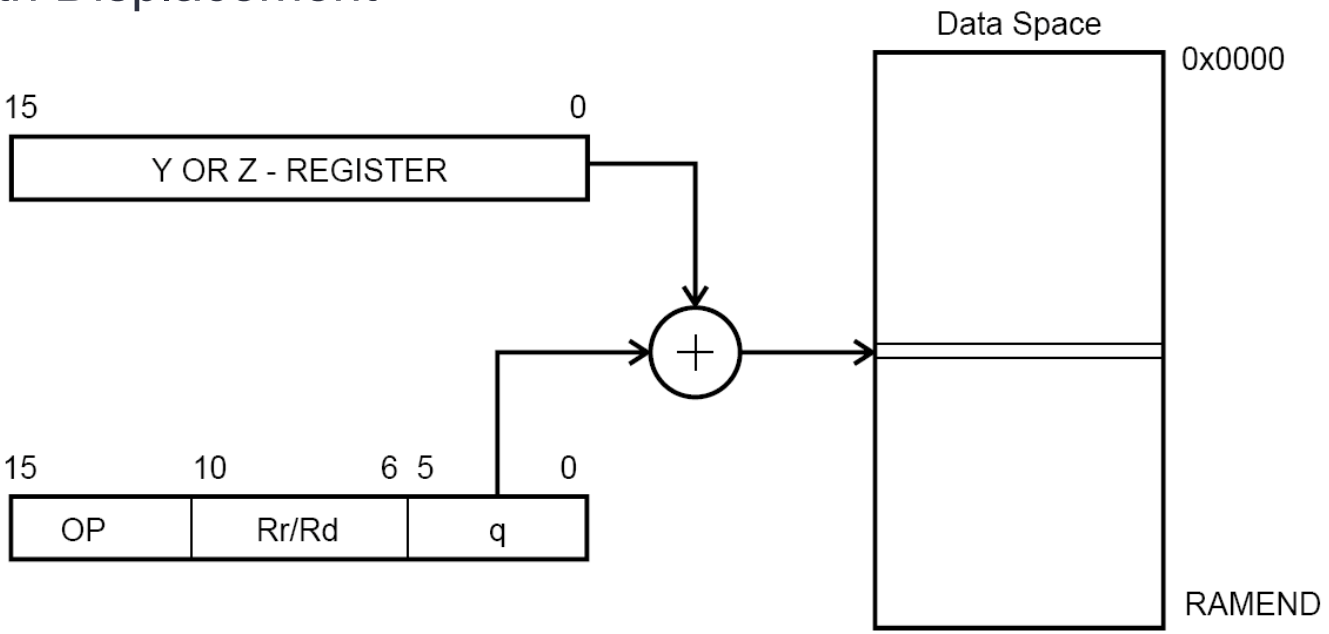


- X,Y,Z = index registers
- RAMEND = hoogste RAM-adres

- Voorbeeld: `ld r16,Z`
- Noot: opcode en register niet weergegeven

Addressing modes

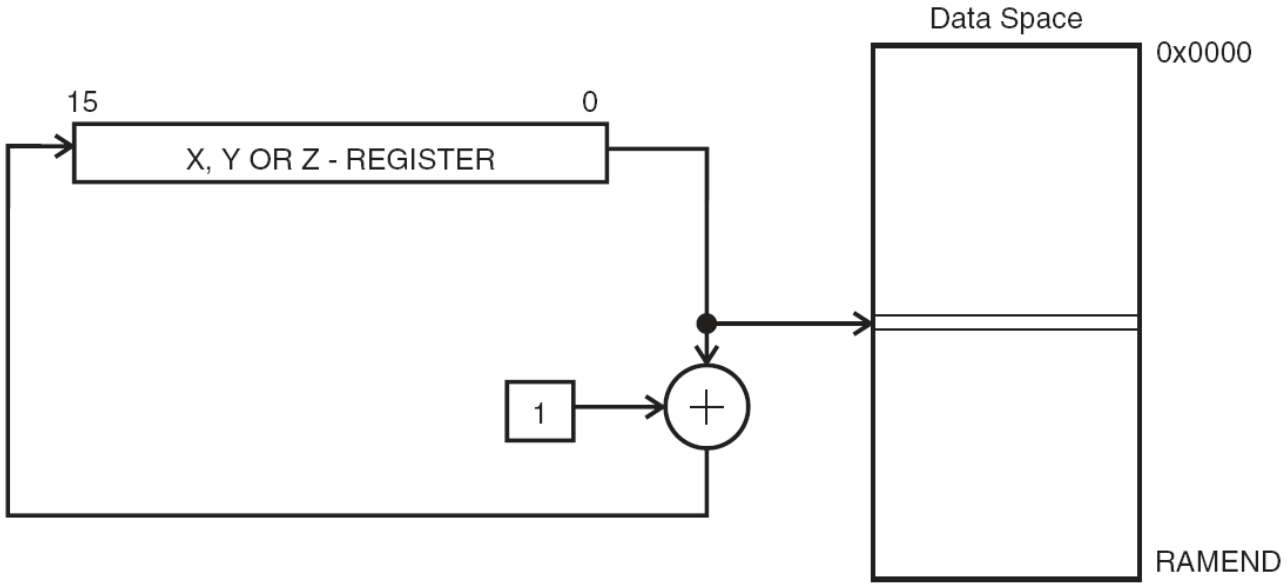
- Data Indirect with Displacement



- q = displacement
- Voorbeeld: `ldd r16,Z+3`
- Noot: alleen Y of Z

Addressing modes

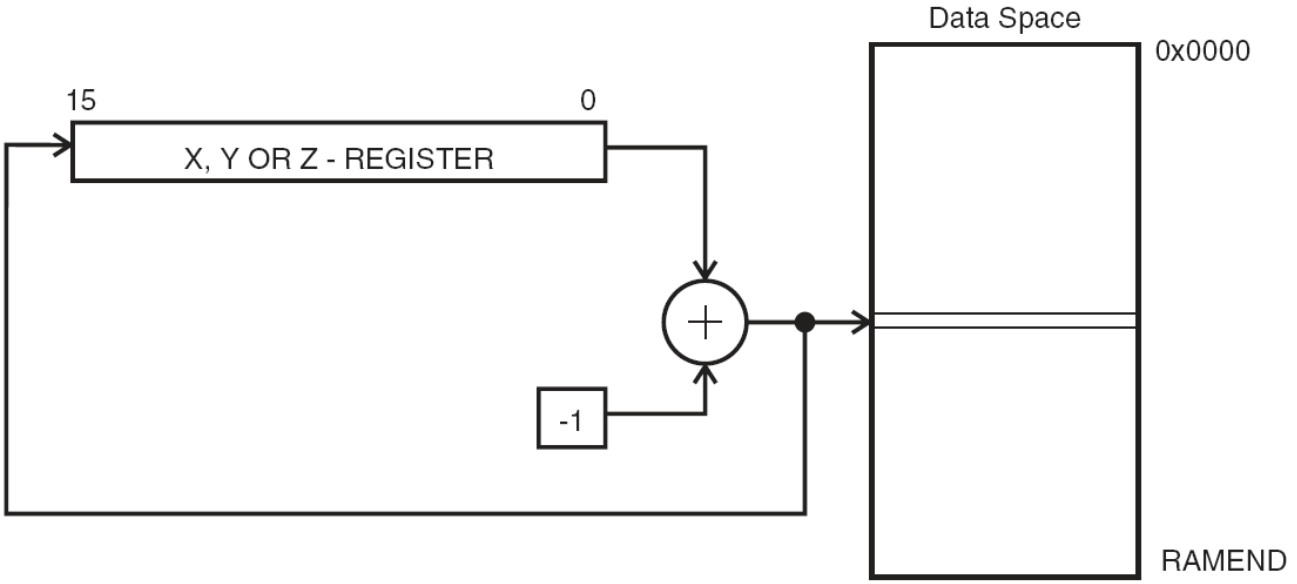
- Data Indirect with Post Increment



- Voorbeeld: `ld r21,Z+`
- Complementair aan Pre Decrement

Addressing modes

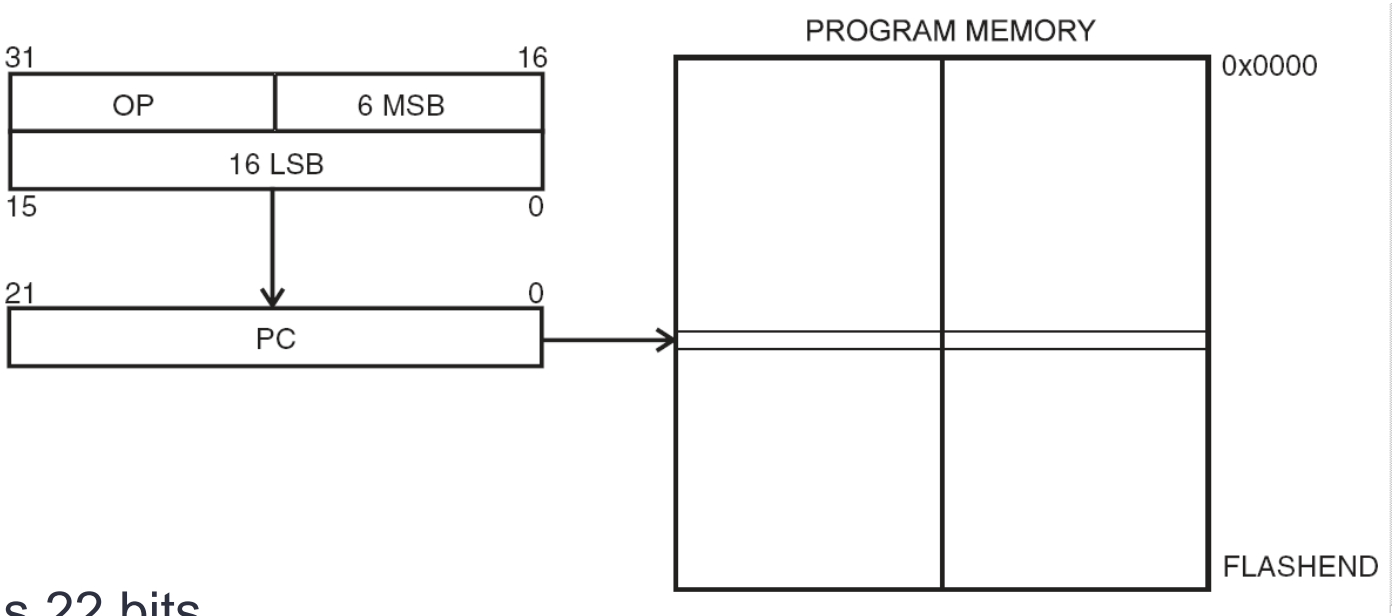
- Data Indirect with Pre Decrement



- Voorbeeld: `ld r12,-Z`
- Complementair aan Post Increment

Addressing modes

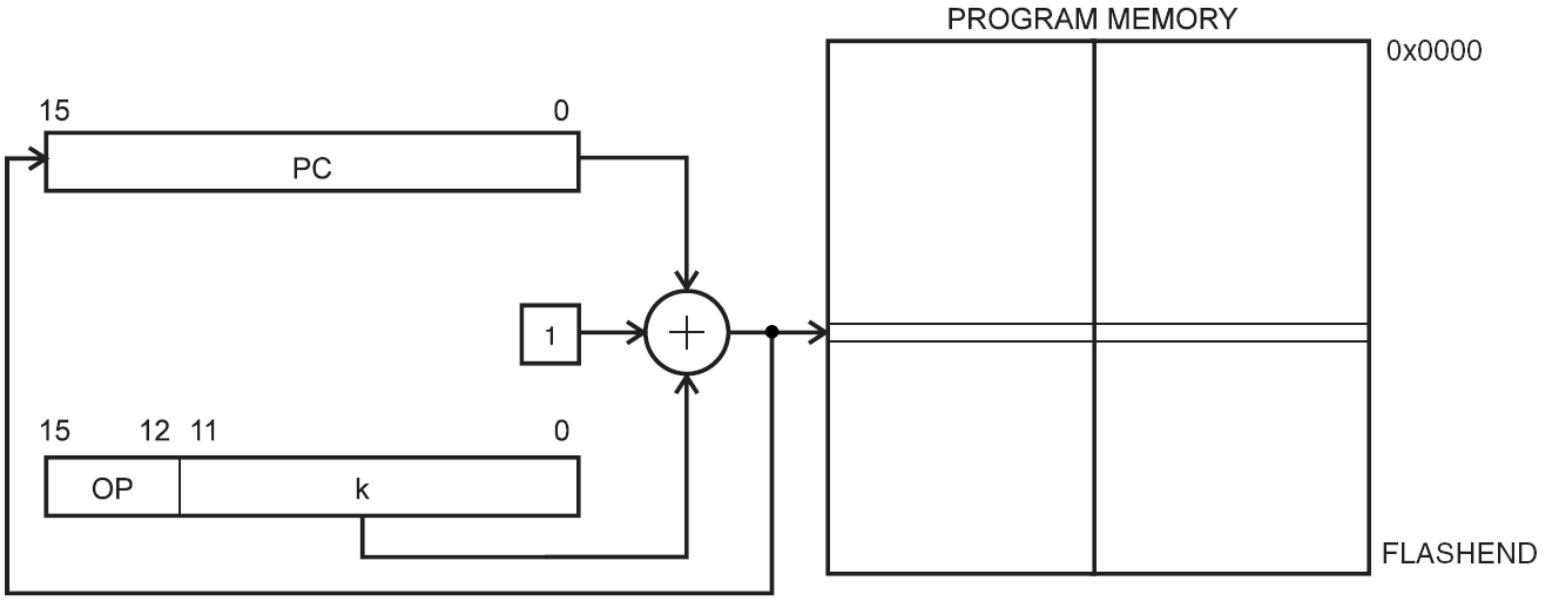
- Direct Program Addressing, JMP en CALL



- Sprongadres is 22 bits.
- FLASHEND = hoogste Flash ROM adres
- Voorbeeld: `call 0x1f0032`

Addressing modes

- Relative Program Addressing, RJMP en RCALL



- Offset is signed, dus vooruit en achteruit springen is mogelijk.
- Handig voor *relocaten* van programma.

Addressing modes

- Er zijn nog wat meer mogelijkheden, deze worden niet besproken. Zie de AVR-documentatie voor meer.

Flags

- Veel instructies passen de flags aan om de toestand van een rekenkundige of logische operatie weer te geven.
- Per instructie opzoeken.
- Let op: `mov`, `ld`, `ldd`, `ldi`, `in` en `out` passen de vlaggen niet aan!
- Nog iets: `inc` en `dec` passen de carry-flag niet aan. Opletten met multi-byte optellingen.

Voorbeeld flags met ADD instructie

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: $N \oplus V$, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Voorbeeld instructie

LDI – Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

Operation:

(i) Rd ← K

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr   r31       ; Clear Z high byte
ldi   r30,$F0   ; Set Z low byte to $F0
lpm             ; Load constant from Program
                ; memory pointed to by Z
```

Words: 1 (2 bytes)
Cycles: 1

DE HAAGSE HOGESCHOOL

Voorbeeld instructie

OR – Logical OR

Description:

Performs the logical OR between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd v Rr

	Syntax:	Operands:	Program Counter:
(i)	OR Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	10rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
        or      r15,r16     ; Do bitwise or between registers
        bst     r15,6       ; Store bit 6 of r15 in T Flag
        brts    ok          ; Branch if T Flag set
        ...
ok:     nop                 ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1

DE HAAGSE HOGESCHOOL

Enige instructies

- mov `mov r1,r3`
- ld,ldd
  ```
  ld r2,Z
  ld r2,Z+
  ld r2,-Z
  ldd r2,Z+5
  ```
- lds `lds r2,0x0060`
- ldi `ldi r16,143`

Enige instructies

- add et al. `add r1,r3` `sub r3,r5, …`
- com et al. `com r1` `neg r5` `inc r6` `dec r9`
- lsl et al. `lsl r3` `lsr r6` `asr r6` `ror r7` `rol r8`
  - `asl` bestaat niet
- jmp et al. `jmp 0x045` `rjmp 0x56`
- andi, ori `andi r24,0x23`
  - `eori` bestaat niet

Enige instructies

- adiw et al. `adiw r31:r30,1` `sbiw r26:r27,5`
- in,out `in r4,0x1b` `out 0x1a,r6`
- subi `subi r16,1`
  truc: `subi r16,-2` telt 2 bij R16 op*.

*) addi bestaat niet.

Voorbeeldcode

```
.nolist
.include "m32def.inc“        ; For ZH and ZL
.list

.def temp   = r16            ; Temporary data
.def index  = r17            ; List index
.def sum    = r18            ; Sum value
.equ list   = 0x0060         ; Start of list in SRAM
.equ length = 8              ; List length

.org 0x000

main:
        ldi  zl,low(list)    ; Z reg points to list
        ldi  zh,high(list)

        ldi  index,1         ; Set index to one
        ld   sum,z           ; Load first in sum
        adiw zh:zl,1         ; Move pointer to next
```

```
while:
        cpi  index,length    ; Did we reach the end?
        brge ewhile          ; If yes, branch out of here

do:
        ld   temp,z          ; Get list[index]
        add  sum,temp        ; Add to sum

        adiw zh:zl,1         ; Move pointer to next
        inc  index           ; Increment index
        rjmp while           ; And do it again

ewhile:

forever:
        rjmp  forever    ; Halt
```

DE HAAGSE HOGESCHOOL

I/O Ports

- We zullen nu de I/O ports gaan gebruiken voor input en output en koppelen aan andere systemen.
- Welke koppelingen zijn er mogelijk?
- Hoe sturen we de individuele bits aan?
- We bespreken: standaard I/O, pull-up en tri-state
- We bespreken bitmaskers.

I/O Ports

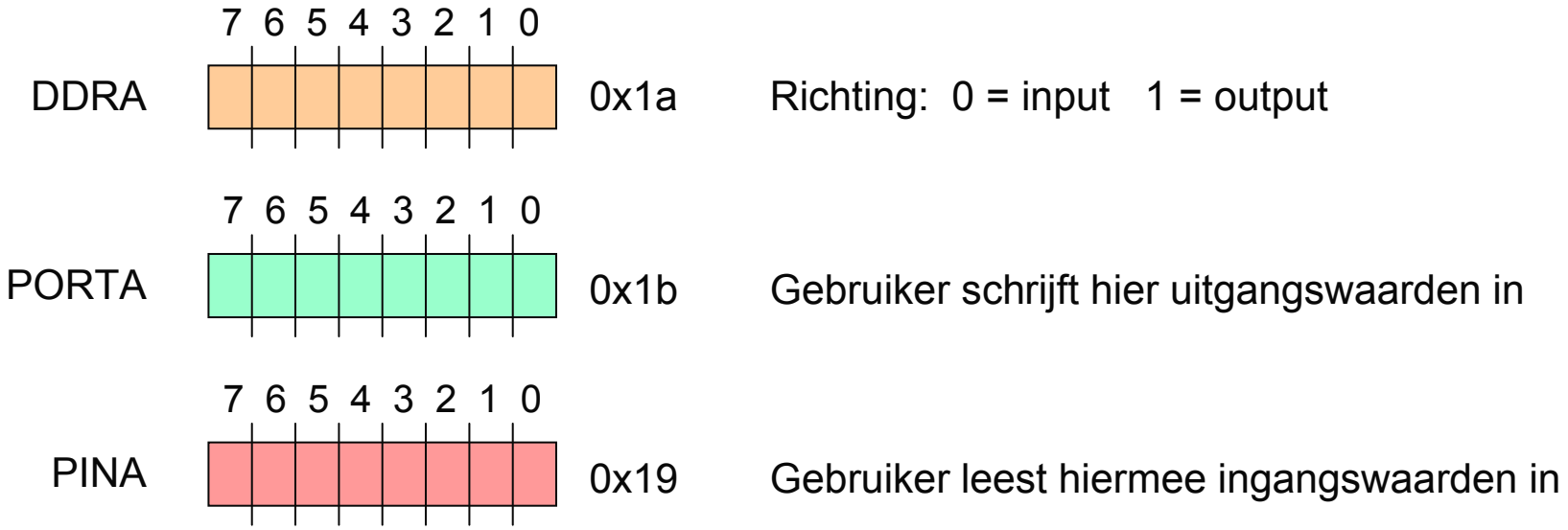
- Om te communiceren met de buitenwereld heeft de ATmega32 een tal van mogelijkheden, onder andere:
  - 32 digitale bi-directionele I/O lijnen.
  - Seriële interface, timers, ADC, ...
- ATmega32 heeft 64 *I/O registers* hiervoor.
- Opmerking: I/O pinnen zijn gemultiplext!

I/O Ports

- De ATmega32 heeft 4 I/O ports van 8 bit per port.
  - PORTA, PORTB, PORTC, PORTD.
- Elke bit is gekoppeld aan een fysieke pin op de behuizing.
  - Voorbeeld: PORTA pin 7 heet PA7.
- Elk bit (pin) kan als ingang of uitgang dienen (*pin-wise controlled*).
- Aansturing gaat per port, dus per 8 bits.
  - Zelf bitmaskers maken voor sturing bits.

I/O Ports

- Per port drie I/O registers: DDR, PORT en PIN
  - DDR = data direction registers, bepaalt de richting.
  - PORT = data dat naar buiten moet gaan (schrijven).
  - PIN = data dat naar binnen moet gaan (lezen).

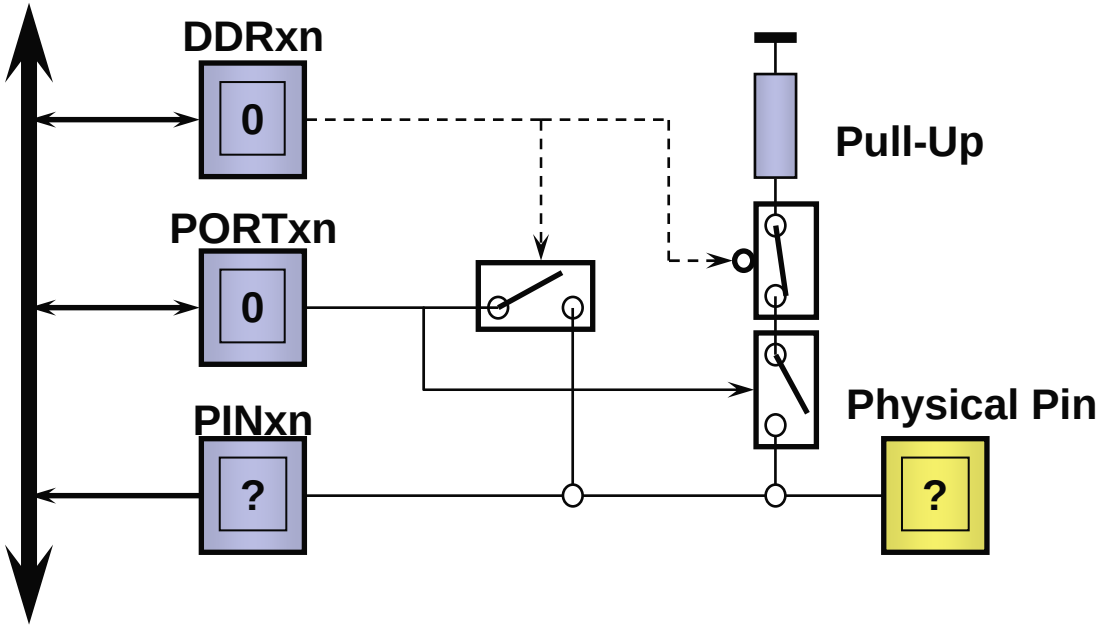


DE HAAGSE HOGESCHOOL

I/O Ports

- Push-Pull Drivers met High Current Drive.
  - Sink/Source tot 20 mA.
- Pull-Up weerstanden.
- Meerdere configuraties.

DDRxn	PORTxn	waarde
0	0	High Z
0	1	Pull-up
1	0	0
1	1	1



I/O ports

PUD = pullup disable

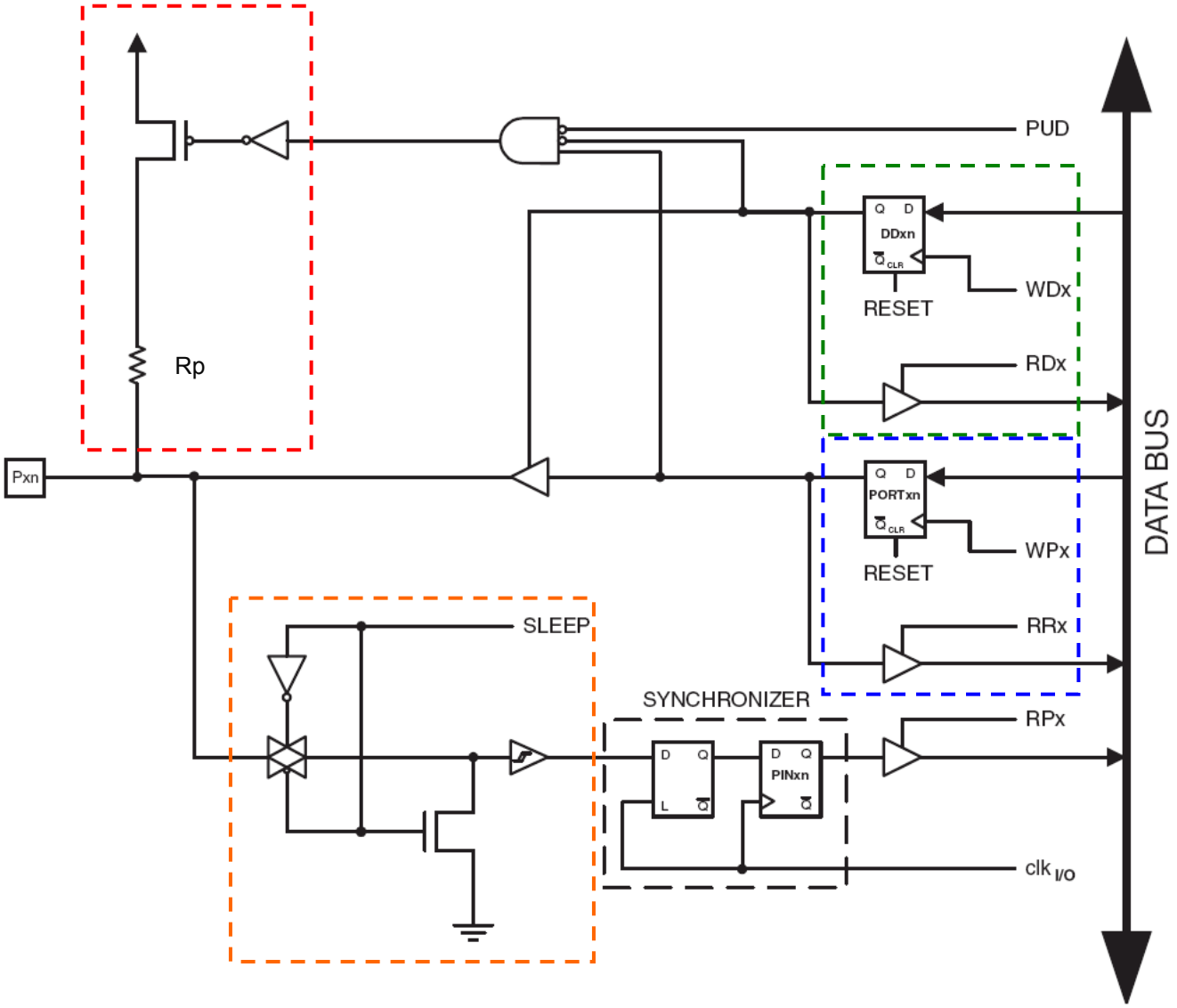
WDx = Write Data
Register bit x

RDx = Read Data
Register bit x

WPx = Write Port
Register bit x

RRx = Read Port
Register bit x

RPx = Read Pin
Register bit x

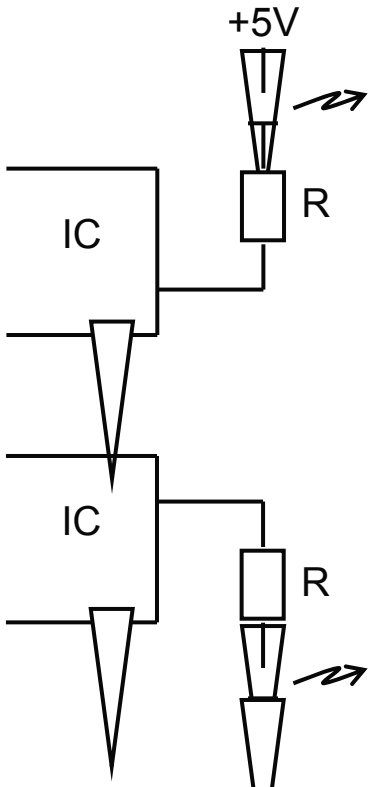


Output

- Hiervoor moet het DDR-bit op 1 gezet worden. Met het PORT-bit kan de juiste waarde worden uitgestuurd.

- Op twee manieren aan te sturen:
  - Active high, active low

- Wat is de waarde van R?
  - Vcc = +5V
  - Vol = 0,4 V
  - Voh = 4,6 V
  - Vled = 1,8 V
  - Iled = 10 mA

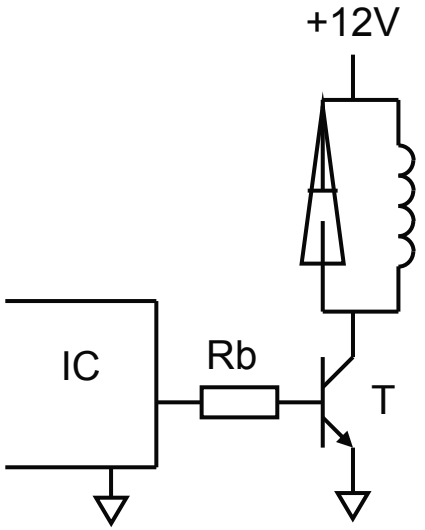


Output

- Aansturen van een spoel:

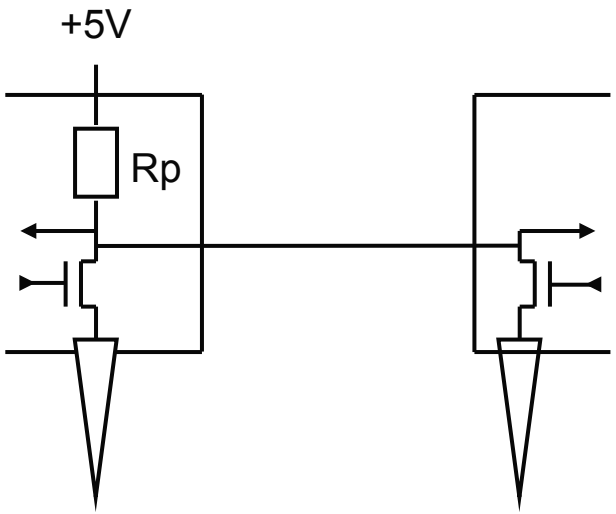
- Wat is de waarde van Rb?
  - Vb = +12V
  - Voh = 4,6 V
  - Ispoel = 23 mA
  - βtor = 100
  - Ube = 0,6 V

- Diode is om inductiespanning te neutraliseren (vrijloopdiode).



Pull-up

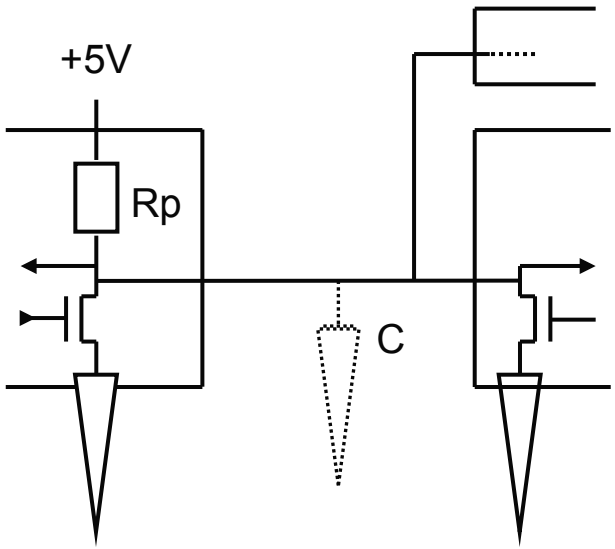
- Met behulp van een pull-up weerstand zijn bi-directionele datalijnen te maken (bijvoorbeeld I^2C).
- Per lijn mag maar één pull-up weerstand zijn*.
- Pull-up weerstand inwendig in AVR, tussen 20 en 50 kΩ.

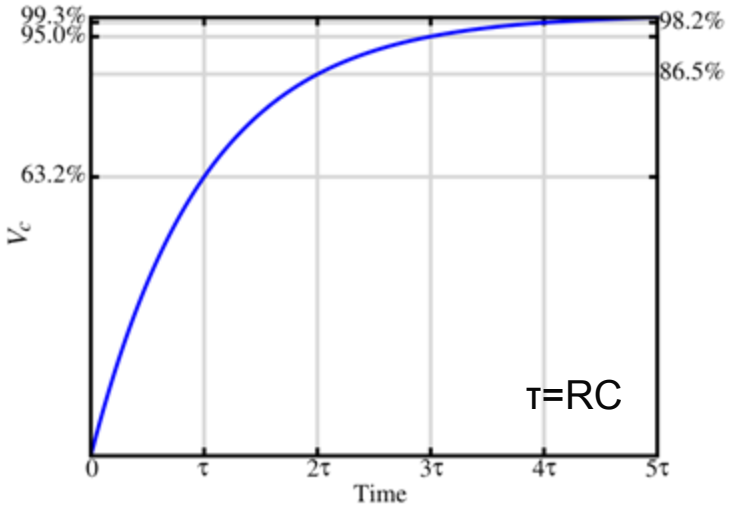


DE HAAGSE HOGESCHOOL

Pull-up

- Voordeel: eenvoudig meerdere bronnen aan te sluiten.
  - Logische 0 overheerst.
- Nadeel: traag herstel van logische 1.
  - Hoge inwendige weerstand.
  - Paracitaire capaciteit.





Pull-up

- Na een reset van de ATMega32 staat de output op High Z.
- Er moet geschakeld worden van logische 0 naar pull-up.
- Let op: Ga nooit via logische 1!

DDRxn	PORTxn	waarde
0	0	High Z
0	1	Pull-up
1	0	0
1	1	1

Tri-state

- Er moet geschakeld worden tussen 0 en 1, 0 en High-Z en 1 en High-Z.

- Schakelen tussen 0 en 1.
  - Alleen PORT aanpassen.

- Schakelen tussen 0 en High-Z.
  - Alleen DDR aanpassen.

- Schakelen tussen 1 en High-Z.
  - Zowel DDR als PORT aanpassen, moet met twee instructies, beste is via Pull-up.

DDRxn	PORTxn	waarde
0	0	High Z
0	1	Pull-up
1	0	0
1	1	1

Aansturing bits

- Probleem: hoe stuur ik nu individuele bits aan?

- Oplossing: gebruik speciale bit-instructies.
  - Werkt niet op alle I/O registers.
  - Niet beschikbaar in C.

- Oplossing: bitmaskers.
  - Werkt op alle I/O registers.
  - Beschikbaar in C.

Bitinstructies

- `cbi` – clear bit in I/O register.
- `sbi` – set bit in I/O register.
- Voorbeeld:

```
sbi PORTA,5    ; set bit 5 in PORTA
cbi DDRA,5     ; clear bit 5 in DDRA
```

- Werkt alleen op de onderste 32 I/O registers.

Bitmaskers

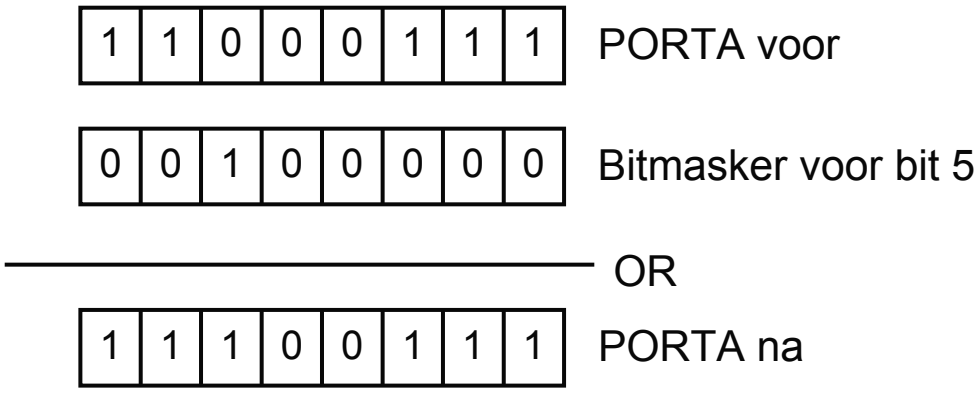
- Met behulp van AND, OR en EXOR kunnen bits in een register worden aangepast.

- Algemene opbouw:
  - Lees I/O register in General Purpose Register
  - Manipuleer bits in General Purpose Register
  - Schrijf General Purpose Register naar I/O register

- OR: zet bits aan.
- AND: zet bits uit.
- EXOR: inverteer bits.

Bitmaskers

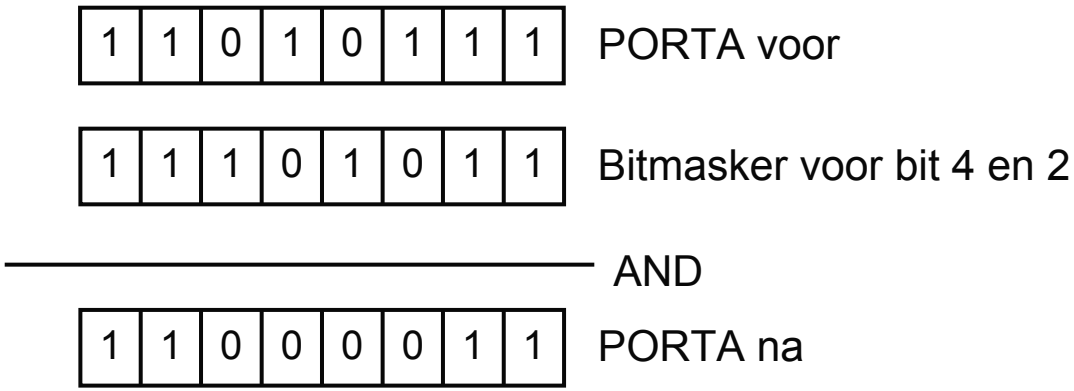
- Aanzetten van bit: OR
  - Alle bits die aan moeten: or-en met 1
  - Alle bits die niet moeten veranderen: or-en met 0




- ```
  in  r16,PORTA
  ori r16,0x20
  out PORTA,r16
  ```
  ```

Bitmaskers

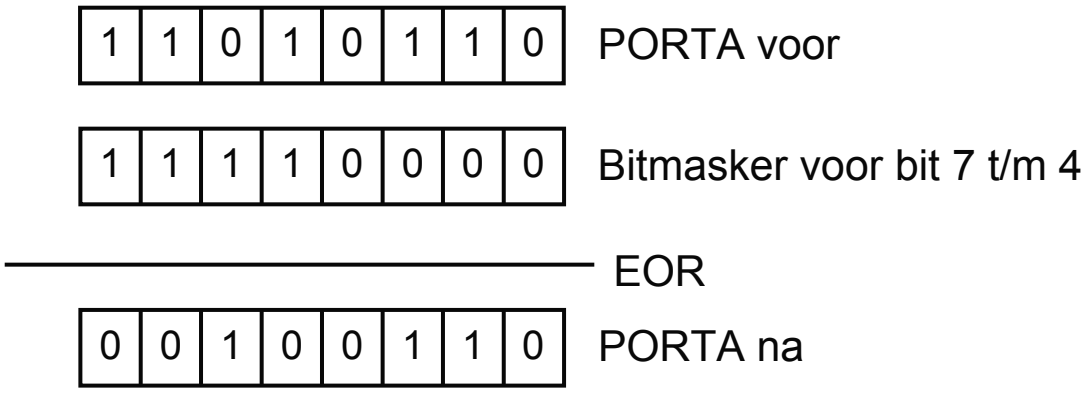
- Uitzetten van bit: AND
  - Alle bits die uit moeten: and-en met 0.
  - Alle bits die niet moeten veranderen: and-en met 1.




- ```
  in   r16,PORTA
  andi r16,0xeb
  out  PORTA,r16
  ```
  ```

Bitmaskers

- Inverteren van bit: EXOR
  - Alle bits die moeten inverteren: eor-en met 1.
  - Alle bits die niet moeten veranderen: eor-en met 0.




- ```
  in  r16,PORTA
  ldi r17,0xf0
  eor r16,r17     ; eori bestaat niet
  out PORTA,r16
  ```
  ```

Beslissen

- Het programmeren van een beslissing valt in twee stukken uiteen:

- Rekenkundige of logische operatie
  - Zet de diverse vlaggen in SREG
  - add, sub, and, or, eor, etc
- Vergelijken (compare):
  - Zet de diverse vlaggen in SREG
  - cp, cpc, cpi, cpse

- Springen (branch)
  - Springen naar een adres als een vlag of een combinatie van vlaggen is gezet (of juist niet).
  - brcc, brcs, brsh, brlo, brne, breq, brpl, brmi, brvc, brvs, brlt, brge, brhc, brhs, brtc, brts, brid, brie.

Rekenkundige of logische operatie

- Bit tegen de linkerkant?

```
        lsl   r1     ; shift highest bit in carry
        brcs  over   ; branch if highest bit was 1
        ...
over:   ...
```

- Springen als teller nog niet op 0 is.

```
        ldi   r16,10 ; load 10
loop:   subi  r23,-5 ; add 5 to r23
        dec   r16    ; decrement r16
        brne  loop   ; is r16 already 0?
```

Vergelijken

- Compare is niets anders dan een subtract-operatie.

  `cp r1,r2` - trek r2 van r1 af en zet de vlaggen.

  `cpi r16,10` - trek 10 van r16 af en zet de vlaggen.

  `cpc r1,r2` - trek r2 en carry van r1 af en zet de vlaggen, wordt gebruikt in multi-byte vergelijkingen.

- In alle gevallen: resultaat aftrekking wordt niet opgeslagen.
- Noot: er bestaat geen `cpci`-instructie.

Vergelijken en beslissen

- Voorbeeld `cp` en `cpi` i.c.m. beslissing:

```
        cp    r1,r2
        breq  over1
        inc   r2
over1:  inc   r1
```

```
        cpi   r1,10
        brne  over2
        sub   r2,1
over2:  add   r2,7
```

- Geheugenplaats testen:

```
        lds   r16,0x0060   ; laad adres 0x0060
        cpi   r16,0        ; 0? (lds zet geen vlaggen)
        breq  over3        ; spring als 0
        ...
```

Beslissen

- Er zijn vier testmogelijkheden:
  - Gelijk aan
  - Ongelijk aan
  - Kleiner
  - Groter of gelijk

- Deze twee zijn niet direct mogelijk met de AVR:
  - Groter
  - Kleiner of gelijk
  - Deze zijn wel na te bootsen

Beslissen

- Kleiner (signed): test de S-vlag.

```
cpi    r16,38
brlt   over5
```

- Kleiner (unsigned): test de C-vlag.

```
cpi    r16,0xf0
brlo   over6
```

- `brlo` is identiek aan `brcs`.

Beslissen

- Groter of gelijk (signed): test de S-vlag.

```
cpi    r16,38
brge   over7
```

- Groter of gelijk (unsigned): test de C-vlag.

```
cpi    r16,0xf0
brsh   over8
```

- `brsh` is identiek aan `brcc`.

Beslissen

- Testen op groter is niet direct mogelijk, want er bestaan geen brgt en brhi. Toch is deze test te doen door de operands om te draaien en te testen op brlt en brlo:

```
cp r1,r2      ; R1 > R2
brgt over9    ; bestaat niet
```

```
cp r2,r1      ; R2 < R1
brlt over9    ; deze wel
```

```
cp    r1,r2
brhi  over10
```

```
cp    r2,r1
brlo  over10
```

- Let op: R1 < R2 ≠ R2 ≥ R1

DE HAAGSE HOGESCHOOL

Beslissen

- Testen op kleiner of gelijk is niet direct mogelijk, want er bestaan geen brle en brls. Toch is deze test te doen door de operands om te draaien en te testen op brge en brsh:

```
cp r1,r2      ; R1 ≤ R2
brle over9    ; bestaat niet
```

```
cp r2,r1      ; R2 ≥ R1
brge over9    ; deze wel
```

```
cp    r1,r2
      brls  over10
```

```
cp    r2,r1
brsh  over10
```

- Let op: R1 ≤ R2 ≠ R2 > R1

Beslissen

- Nog een paar over:

`brmi`	-	branch on minus (N=1)
`brpl`	-	branch on plus (N=0)
`brhc`	-	branch on half carry clear (H=0)
`brhs`	-	branch on half carry set (H=1)
`brtc`	-	branch on T-flag clear (T=0)
`brts`	-	branch on T-flag set (T=1)
`brvc`	-	branch on overflow clear (V=0)
`brvs`	-	branch on overflow set (V=1)

Conditional Branch Summary

Test	Boolean	Mnemonic	Complementary	Boolean	Mnemonic	Comment
Rd > Rr	Z•(N ⊕ V) = 0	BRLT(1)	Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE*	Signed
Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Rd < Rr	(N ⊕ V) = 1	BRLT	Signed
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Signed
Rd ≤ Rr	Z+(N ⊕ V) = 1	BRGE(1)	Rd > Rr	Z•(N ⊕ V) = 0	BRLT*	Signed
Rd < Rr	(N ⊕ V) = 1	BRLT	Rd ≥ Rr	(N ⊕ V) = 0	BRGE	Signed
Rd > Rr	C + Z = 0	BRLO(1)	Rd ≤ Rr	C + Z = 1	BRSH*	Unsigned
Rd ≥ Rr	C = 0	BRSH/BRCC	Rd < Rr	C = 1	BRLO/BRCS	Unsigned
Rd = Rr	Z = 1	BREQ	Rd ≠ Rr	Z = 0	BRNE	Unsigned
Rd ≤ Rr	C + Z = 1	BRSH(1)	Rd > Rr	C + Z = 0	BRLO*	Unsigned
Rd < Rr	C = 1	BRLO/BRCS	Rd ≥ Rr	C = 0	BRSH/BRCC	Unsigned
Carry	C = 1	BRCS	No carry	C = 0	BRCC	Simple
Negative	N = 1	BRMI	Positive	N = 0	BRPL	Simple
Overflow	V = 1	BRVS	No overflow	V = 0	BRVC	Simple
Zero	Z = 1	BREQ	Not zero	Z = 0	BRNE	Simple

Note: 1. Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd

Multi-byte vergelijken

- Let op hoe de twee aftrekinstructies de Z- en C-vlaggen beinvloeden:

`subi`	`reg,const`	C = 1 als unsigned resultaat < 0, anders 0
`sub`	`reg,reg`	Z = 1 als resultaat = 0
		Z = 0 als resultaat ≠ 0
`sbci`	`reg,const`	C = 1 als unsigned resultaat < 0, anders 0
`sbc`	`reg,reg`	Z = 0 als resultaat ≠ 0

- De `sbci`- en `sbc`-instructies kunnen de Z-flag alleen op 0 zetten, niet op 1. Hierdoor zijn multi-byte vergelijkoperaties mogelijk.

Multi-byte vergelijken

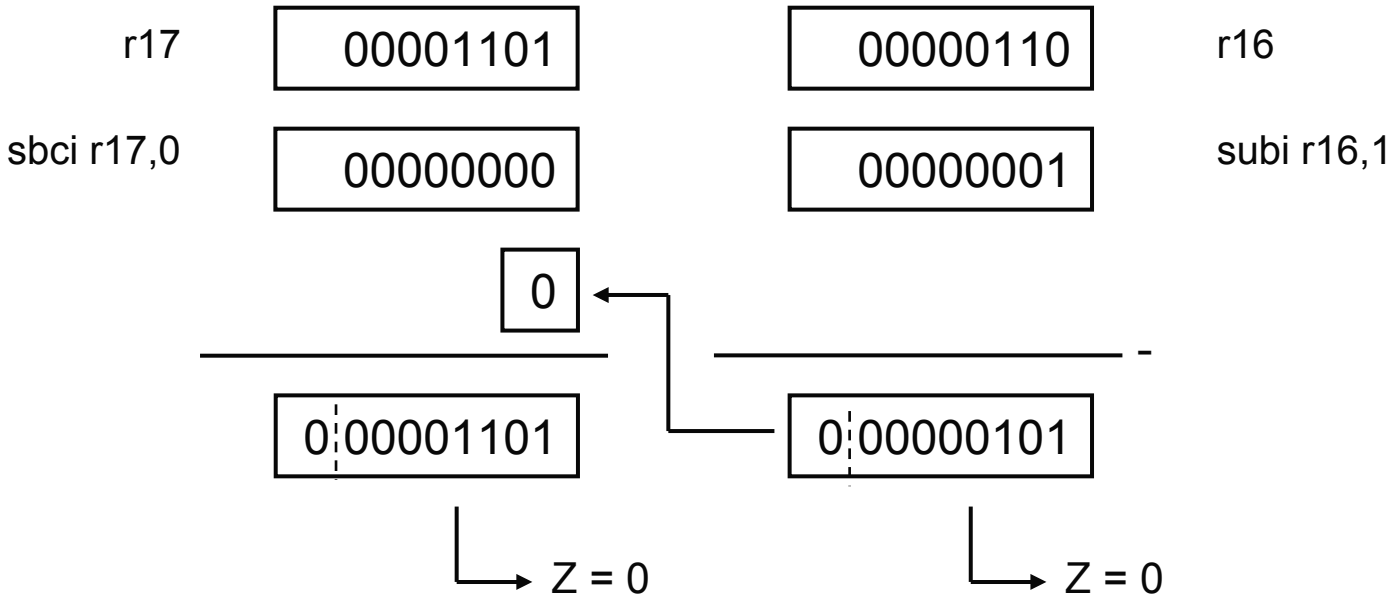
- Stel dat we een 16-bits variabele, opgeslagen in registers R17 en R16 met 1 willen verlagen én testen of het resultaat gelijk is aan 0.

- Dat kan met de volgende instructie-reeks.

```
        subi   r16,0x01
        sbci   r17,0x00
        breq   label
        ...
label:
```

Multi-byte vergelijken

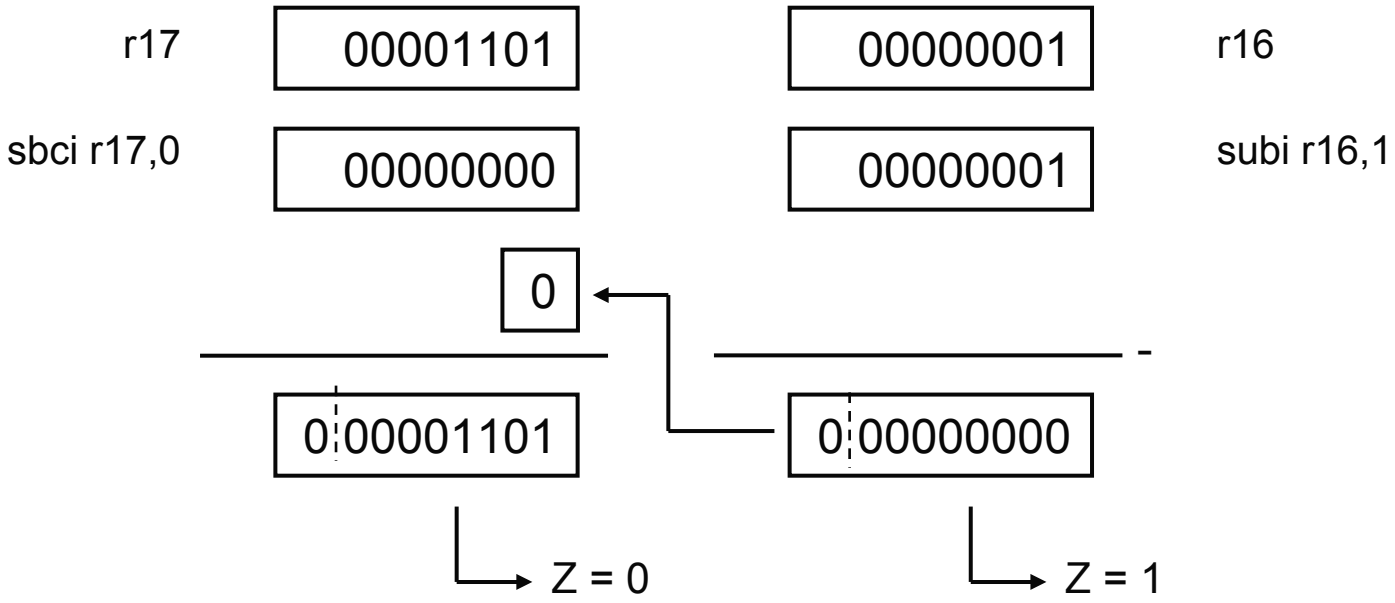
- R16 ≠ 1 en R17 ≠ 0



- Resultaat R17:R16 ≠ 0. De `sbci`-instructie zet de Z-flag op 0.

Multi-byte vergelijken

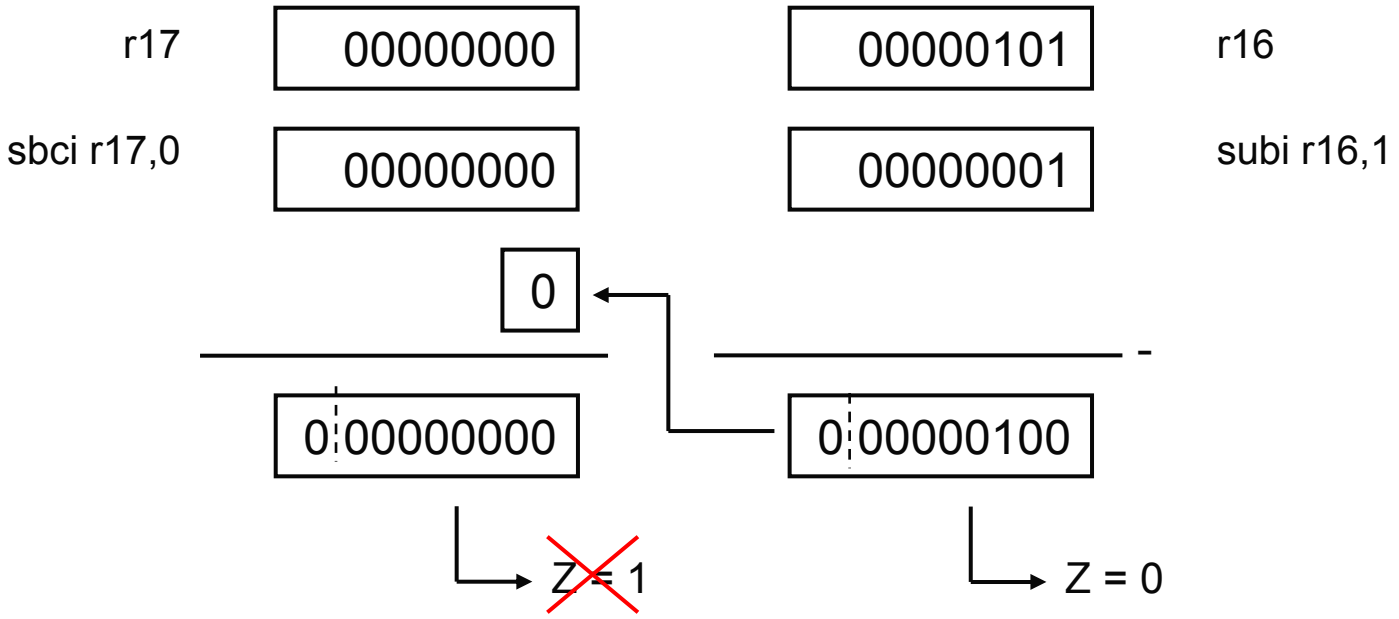
- R16 = 1 en R17 ≠ 0



- Resultaat R17:R16 ≠ 0. De `sbci`-instructie zet de Z-flag op 0.

Multi-byte vergelijken

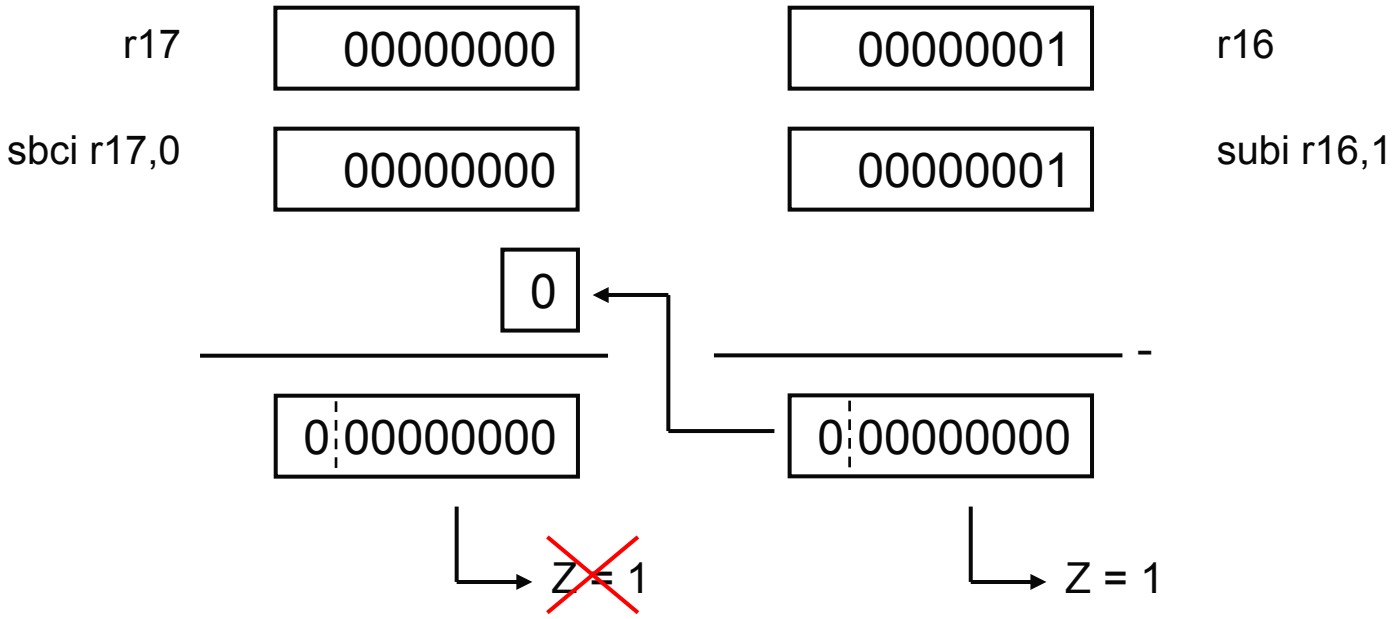
- R16 ≠ 1 en R17 = 0



- Resultaat R17:R16 ≠ 0. De `sbci`-instructie zet de Z-flag ***niet*** op 1. Netto resultaat: Z = 0.

Multi-byte vergelijken

- R16 = 1 en R17 = 0



- Resultaat R17:R16 = 0. De `sbci`-instructie zet de Z-flag ***niet*** op 1. Netto resultaat: Z = 1.

Lussen

- Er zijn twee soorten lussen:
  - Vast aantal: doe 10 keer ….
  - Variable aantal: doe zolang bitje is 1 …
- Vast aantal: gebruik een teller die steeds met 1 verhoogd wordt totdat het aantal keer bereikt is.
- Variabel aantal: doe een test en blijf wachten.

Vast aantal lus

- Twee manieren: teller verhogen of verlagen.
- Probleem bij verlagen: brge en geen brgt.

```
; teller van 0 naar 9
        ldi  r16,0
loop:   ...
        inc  r16
        cpi  r16,10
        brlt loop
```

```
; teller van 9 naar 0
        ldi  r16,9
loop:   ...
        dec  r16
        cpi  r16,0    *)
        brge loop
```

*) mag weggelaten worden, dec past de Z-flag aan.

Variabel

- Wacht op een bepaalde conditie.

- Voorbeeld: wacht tot bit 5 van Port A 1 is.
- Voorbeeld: wacht tot ADCH-waarde groter/gelijk aan 45.

```
        ...
loop:   in     r16,PINA
        andi   r16,0x20
        breq   loop:  *)
        ...
```

```
        ...
loop:   in     r16,ADCH
        cpi    r16,45
        brlt   loop
        ...
```

*) `andi` past Z-flag aan.

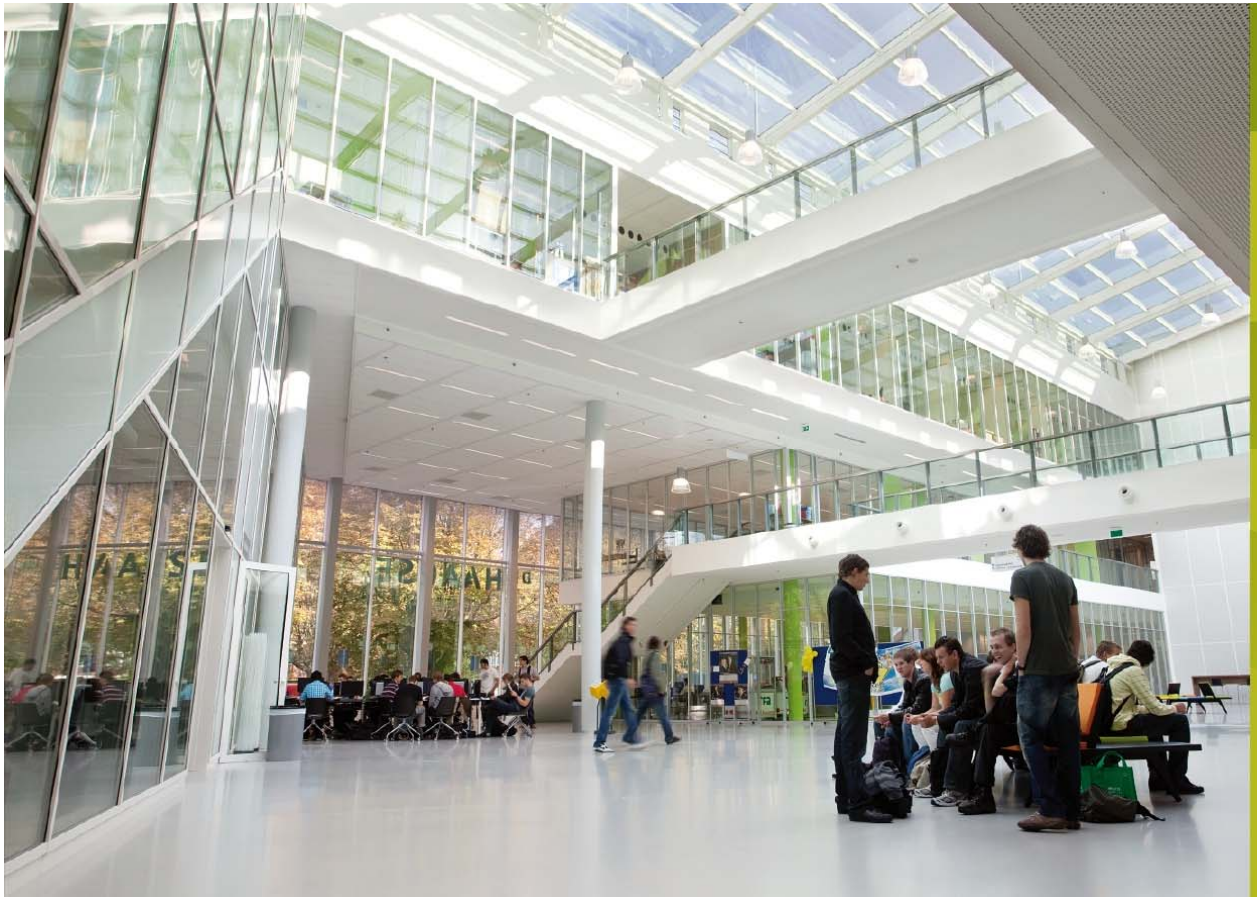
Lopen langs RAM-geheugen

- Bij het lopen langs RAM-geheugen moeten we de Z-index gebruiken.

```
        ldi   r30,0x60     ; pointer naar 0x0060
        ldi   r31,0x00     ; R31:R30 is Z-index
        ldi   r16,0        ; teller
loop:   ld    r17,Z+       ; ophalen gegeven
        ...                ; doe iets
        inc   r16          ; teller verhogen
        cpi   r16,10       ; is het klaar
        brlt  loop         ; nee, dan nog een keer
```

Literatuur

- H3S3, H3S5, H3S6, H3S7, H4S1, H6S1, H6S2, H6S3, H6S4
- Amerikaanse editie: H2S3, H2S5, H2S6, H2S7, H3S1, H5S1, H5S2, H5S3, H5S4



Academie voor Technology, Innovation & Society Delft

Academie voor ICT & Media

De Haagse Hogeschool, Delft
015-2606311
J.E.J.opdenBrouw@hhs.nl
www.dehaagsehogeschool.nl

DE HAAGSE HOGESCHOOL